Deliverable D1.3:

NEMO Tangent & Adjoint Models (NemoTam)

Reference Manual & User's Guide

Arthur Vidard, Franck Vigilant
INRIA/LJK, Grenoble

Charles Deltel, Rachid Benshila
LOCEAN-IPSL, Paris

VODA: ANR-08-COSI-016

Contents

1 NEMOTAM REFERENCE MANUAL **5**

1.1 Introduction . . . 5

1.1.1 Abstract . . . 5

1.1.2 Introduction . . . 5

1.2 Tangent and adjoint models (TAM) basics . . . 7

1.2.1 TAM, but what for ? . . . 7

1.2.2 Tangent and Adjoint coding principle . . . 7

1.2.3 Potential issues . . . 8

1.3 Direct model changes . . . 8

1.3.1 Storing the non-linear trajectory . . . 8

1.3.2 Modified routines in the direct code . . . 10

1.3.3 Modified declarations in the direct code . . . 10

1.4 Tangent linear model approximations . . . 10

1.4.1 Active variables degraded to passive . . . 10

1.4.2 Simplifications . . . 11

1.5 Hand coding status . . . 11

1.6 Available options in NEMOTAM . . . 11

1.7 The generic validation interface . . . 16

1.7.1 Introduction . . . 16

1.7.2 Adjoint test . . . 16

1.7.3 Tangent-Linear validation . . . 17

1.7.4 Tangent-Linear Hypothesis (HLT) validity . . . 19

1.8 MPI parallelization . . . 20

1.8.1 General features . . . 20

1.8.2 Adjoint of MPI_ALLGATHER . . . 21

2 NEMOTAM USER'S GUIDE **23**

2.1 Source code structuration . . . 23

2.2 Getting started tutorial . . . 24

2.2.1 Installing the code using the NEMOVAR GIT repository . . . 24

2.2.2 Installing the code using the NEMOTAM SVN repository . . . 25

2.2.3 Which executable for which test? . . . 26

2.3 Demonstrators . . . 26

2.3.1 The ORCA2 global configuration . . . 26
2.3.2 The GYRE regional configuration . . . 32
2.3.3 The POMME configuration (toy model for handling open boundaries) . . . 33
2.4 How-to update NEMOTAM with your contribution? . . . 34
2.4.1 Coding rules for TAM . . . 34
2.4.2 Workflow for introducing additional developments . . . 35
2.5 Frequently asked questions . . . 36
2.5.1 I can't use GDB after compiling with FCM . . . 36
2.5.2 I have many commas in the namelist generated by piano . . . 36
2.5.3 I have negative values in my cost function terms . . . 36
2.5.4 My run stops with 'PROBLEM WITH THE MATRIX OR WITH THE PRECONDITIONER' on stderr . . . 36

Chapter 1

NEMOTAM REFERENCE MANUAL

1.1 Introduction

This document still contains empty parts. This is done on purpose since it will be the basis for the next deliverable (D1.3 NEMOTAM Manual). We will fill them in as we proceed.

1.1.1 Abstract

The development of the tangent linear and adjoint models (TAM in the following) of the dynamical core of the NEMO ocean engine (NEMOTAM) is a key objective of the VODA project. TAM are widely used for variational assimilation applications, but they are also powerful tools for the analysis of physical processes, since they can be used for sensitivity analysis, parameter identification and for the computation of characteristic vectors (singular vectors, Liapunov vectors, etc.).

In the framework of VODA, a work package has been set-up in order to develop a comprehensive NEMOTAM package, and to define an effective long-term development strategy for ensuring synchronisation of NEMOTAM with future NEMO releases. This is a heavy task, but it is worth the effort since NEMOTAM will benefit all NEMO users for the wide range of applications described above.

Ideally, this strategy should be defined to allow NEMOTAM to adapt to future NEMO developments as quickly and as efficiently as possible, so that new releases of NEMOTAM can be made soon after new releases of NEMO. This will require careful coordination between the main development teams of NEMO, NEMOTAM and possibly NEMOVAR (INRIA, NEMO Team, CERFACS, ECMWF).

1.1.2 Introduction

The NEMO ocean engine (**?**) was previously known as the OPA model (**?**). It used to have a TAM (called OPATAM), fully hand-coded and maintained mainly by A. Weaver. OPATAM was initially developed for a Pacific ocean configuration, and targeted at variational data assimilation applications in the framework of OPAVAR (**??**). OPATAM/OPAVAR were extended to other regional basins (Mediterranean sea (**?**), North Atlantic 1/3°(**?**), South Atlantic 1°), to the global

ocean (ORCA 2° (**?**)), and were used for methodological studies such as control of the 3D model error (**?**), control of the surface forcing and open boundary conditions (**??**). OPATAM was also used for sensitivity studies (**?**), singular vectors (**??**), etc.

For several reasons, mainly because of lack of workforce, OPATAM, OPAVAR and related developments were not included in the standard release of OPA. As a consequence, synchronisation of OPATAM with OPA's releases could not be achieved on a regular basis, and all developments were on individual branches, without feedback to the OPATAM/OPAVAR system. The pool of potential users was reduced consequently. It is important not to repeat this error in the future, so as to ensure that NEMOTAM become a widely used community tool.

A NEMOTAM working group was initiated in the framework of a CNRS/INSU/LEFE ASSIM-2006 project, to investigate the feasibility of using TAPENADE (**?**), an AD tool, to speed up the writing of TAM for OPA9, the dynamical ocean component of NEMO. The goal of this working group was twofold. The first goal was to identify the strengths and weaknesses of TAPENADE by applying it directly to the NEMO source code, with as little human intervention as possible, in order to build a NEMOTAM prototype. The second goal was to define, based on the experience deriving the prototype, a strategy for developing a general-purpose NEMOTAM that both respects the NEMO code style/structure and gives acceptable computer performance (in terms of both CPU and memory requirements) for realistic ocean configurations. Providing feedback to the TAPENADE developers was an important aspect of this work to help improve future versions of the TAPENADE software.

The results from this feasibility study demonstrated that TAPENADE was able to produce tangent-linear and adjoint models for NEMO, albeit for a fixed and somewhat simplified configuration (ORCA 2°with all non-differentiable options switched off (**?**)). Even for this simplified configuration, however, substantial human intervention and additional work was required to obtain a useable product from the raw TAPENADE-generated code. Three main drawbacks with TAPENADE were identified for this application. First, the memory management and CPU performance of the raw code were rather poor. Second, the current version of TAPENADE generates single-processor code only and cannot handle directives from the C-PreProcessor (CPP keys), which are widespread in NEMO. Third, the technique of binomial checkpointing that is used in TAPENADE to handle nonlinearities (see (**?**)) is not compatible, at least in its present implementation, with the incremental algorithm of NEMOVAR, which employs separate executables and (possibly) different resolutions for the outer and inner loops. Improved memory management and extensions to support MPP and CPP keys are planned in future versions of TAPENADE so the first two deficiencies are not fundamental. The third deficiency, however, is more problematic and it is likely that the trajectory management for nonlinearities in NEMOTAM will be done differently from TAPENADE, possibly along the lines of the simpler strategy implemented in OPAVAR. The modifications required to make TAPENADE or whatever other AD tool, compatible with the multi-incremental approach are really substantial and cannot be done in a short or medium term. Moreover the numerical performances of the TAPENADE generated TAM do not allow yet their use for 'big' configurations and for operational applications.

From that experience it has been decided to go toward the hand-coding approach, the use of AD tool being left aside for the time being. We may reconsider this in a medium or long term though.

1.2 Tangent and adjoint models (TAM) basics

1.2.1 TAM, but what for ?

The development of tangent and adjoint models is an important step in addressing sensitivity analysis and variational data assimilation problems in Oceanography. Sensitivity analysis is the study of how model output varies with changes in model inputs. The sensitivity information given by the adjoint model is used directly to gain an understanding of the physical processes. In data assimilation, one considers a cost function which is a measure of the model-data misfit. The adjoint sensitivities are used to build the gradient for descent algorithms. Similarly the tangent model is used in the context of the incremental algorithms to linearize the cost function around a background control.

1.2.2 Tangent and Adjoint coding principle

The original program P, whatever its size and run time, computes a function

$$F : X \in \mathbb{R}^m \rightarrow Y \in \mathbb{R}^n$$

which is the composition of the elementary functions computed by each run-time instruction. In other words if P executes a sequence of elementary statements $I_k, k \in [1, .., p]$, then P actually evaluates

$$F = f_p \circ f_{p-1} \circ ... \circ f_1$$

where each f_k is the function implemented bu I_k. Therefore one can apply the chain rule of derivative calculus to get the Jacobian matrix F', i.e. the partial derivatives of each component of Y with respect to each component of X. Calling $X_0 = X$ and $X_k = f_k(X_{k-1})$ the successive values of all intermediate variables, i.e. the successive states of the memory throughout execution of P, we get

$$F'(X) = f'_p(X_{p-1}) \times f'_{p-1}(X_{p-2}) \times ... \times f'_1(X_0) \tag{1.1}$$

The derivatives f'_k of each elementary instruction are easily built, and must be inserted in the differentiated program so that each of them has the values X_{k-1} directly available for use. This process yields analytic derivatives, that are exact up to numerical accuracy. In practice, two sorts of derivatives are of particular importance in scientific computing: the tangent (or directional) derivatives, and the adjoint (or reverse) derivatives. The tangent derivative is the product

$$dY = F'(X) \times dX$$

of the full Jacobian times a direction dX in the input space. From equation (1.1), we find

$$dY = F'(X) \times dX = f'_p(X_{p-1}) \times f'_{p-1}(X_{p-2}) \times ... \times f'_1(X_0) \times dX \tag{1.2}$$

which is most cheaply executed from right to left because matrix $\times$ vector products are much cheaper than matrix $\times$ matrix products. This is also the most convenient execution order because it uses the intermediate values X_k in the same order as the program P builds them. On the other

hand the adjoint derivative is the product $X_{ad} = F'^{\star}(X) \times Y_{ad}$ of the transposed Jacobian times a weight vector Y_{ad} in the output space. The resulting X_{ad} is the gradient of the dot product $(Y.Y_{ad})$. From equation (1), we find

$$X_{ad} = F'^{\star}(X) \times Y_{ad} = f_1'^{\star}(X_0) \times ... \times f_{p-1}'^{\star}(X_{p-2}) \times f_p'^{\star}(X_{p-1}) \times Y_{ad} \tag{1.3}$$

which is also most cheaply executed from right to left. However, this uses the intermediate values X_k in the inverse of their building order in P.

For specific details about practical adjoint coding refer to (**?**).

1.2.3 Potential issues

- the major approximations : non differentiability issues, simplification of the direct model before linearization, validity of the tangent linear hypothesis over some time window, etc.
- the validation interface : classical tests for checking the correctness of tangent and adjoint models
- numerical problems
- numerical cost issues

1.3 Direct model changes

Some modifications and additions are compulsory to integrate the Tangent and Adjoint Module to NEMO. They mainly consist in:

- changing subroutines or parameters declaration from `private` to `public` for:
  - initialization
  - re-use
- adding a subroutine to save the non-linear trajectory
- adding parameters
- updating namelist

1.3.1 Storing the non-linear trajectory

To evaluate the tangent and/or the adjoint at a time t, some actual parameters of the direct model are needed at the corresponding time t. To fulfill this purpose a subroutine, tamtrj_wri is added to write the trajectory. It is controlled with one logical parameter: ln_trjwri. The trajectory is saved upon a given frequency "nittrjfrq" only if the logical parameter is set to TRUE.

New subroutine tam_trj_wri

A new subdirectory TAM is created into OPA_SRC named TAM. The file is tamtrj.F90. The purpose of this routine is to write on disk the model state trajectory for use with 4DVar. It is called in the `opa_model` routine for the initializiation, and in the `stp` routine in order to perform the actual saving of the trajectory. The currently stored fields are listed in Table 1.1.

Parameters	Associated ccp keys
emp	
emps	
un	
vn	
tn	
sn	
ta	
sa	
tb	
sb	
avmu	
avmv	
aeiu	lk_traldf_eiv
aeiv	lk_traldf_eiv
aeiw	lk_traldf_eiv
uslp	key_ldfslp
vslp	key_ldfslp
wslpi	key_ldfslp
wslpj	key_ldfslp
avs	key_zdfddm
strdmp	key_tradmp
hmlp	key_tradmp

Table 1.1: Fields stored in the reference trajectory

Specific management of the initial state

For the initial state of the model, we call the tam_trj_wri in the 'step' routine just before the tracer block. We, then, have access to some parameters (tke and sbc for instance) as they are not yet computed in opa (outside 'step').

Linear interpolation

For efficiency matters, the whole trajectory is of course not saved. Only a predefined subset of the time steps is saved. The frequency on which the trajectory is saved is controlled by namelist parameter `nittrjfrq`. The intermediate steps will be estimated in TAM by the mean of a linear interpolation.

Namelist

Parameters related to the model state saving is controlled by the namelist `namtam` with:

- ln_trjwri: Logical switch for writing out state trajectory
- nittrjfrq: saving frequency of the trajectory

1.3.2 Modified routines in the direct code

- opa.F90:
  - include initilialization of the module that handles the trajectory saving (`USE tamtrj`)
- step.F90:
  - include calls for saving the trajectory (`CALL tam_trj_wri`)
  - add temporary variables (zta, zsa) to save ta/sa (this is a temporary fix because these variables are currently used as workspace in dynamic block)

1.3.3 Modified declarations in the direct code

The following table describes the change from PRIVATE to PUBLIC of some routines and parameters.

1.4 Tangent linear model approximations

1.4.1 Active variables degraded to passive

Some active variables are treated as passive (mainly due to non-differentiability characteristics).

- avmu, avmv: the vertical eddy viscosity. The turbulent closure scheme `zdftke` induces some non-differentiabilities for there variables. They are used in routine `dynzdf_imp_tam.F90`.

Modules	Routines	Parameters
dynadv	dyn_adv_ctl	
daymod	day_mth	
lib_mpp		mpi_comm_opa
par_oce		jpni, jpnj, jpnij
zdftke	tke_rst	
eosbn2		neos_init
tradmp	cofdis, dtacof, dtacof_zoom	
		strdmp, ttrdmp, resto
traqsr		nksr, gdsr

Table 1.2: List of modified declarations in the direct code NEMO

- uslp, vslp: the slope of neutral surface (slope of isopycnal surfaces referenced locally) are treated as passive in order to not recomputed them at each step. They are used in routine `traldf_iso_tam.F90`.
- wslpi/j: vertical mixing coefficient due to lateral mixing. They are used in routine `trazdf_imp_tam.F90`.

1.4.2 Simplifications

- removal of the upstream part of the second order centered scheme for `traadv_cen2_tam.F90`. It introduces some non-differentiabilities. Some complementary test must be held to fully implement the second order scheme
- removal of zonal mean lateral diffusive heat and salt transport (pht_ldf and pst_ldf)

1.5 Hand coding status

The current status for the hand coding part of NEMOTAM / NEMOVAR is gathered in Tables 1.3 and 1.4.

1.6 Available options in NEMOTAM

This sections contains information on current available configuration for NEMOTAM. To have an overview on available routines, please see the TAM development progress table. We use the following method: The physical choices are set through cpp keys and namelist parameters. We use below a similar table used for NEMO. See tables 1.5 and 1.6.

Modules called by step_tam	Module(s) called by previous one	Options	Remarks
Ocean dynamics (DYN)			
cla_div_tam.F90			
cla_tam.F90			
divcur_tam.F90			
dynadv_tam.F90	dynkeg_tam.F90 dynzad_tam.F90 dynadv_cen2_tam.F90 dynadv_ubs_tam.F90		not in reference not in reference
dynhpg_tam.F90		hpg_zco_hpg_zps hpg_sco_hpg_hel hpg_wdj, hpg_djc, hpg_rot	
dynldf_tam.F90	dynldf_lap_tam.F90 dynldf_bilap_tam.F90 dynldf_iso_tam.F90 dynldf_bilapg_tam.F90		not in reference not in reference
dynnxt_tam.F90		lk_vvl key_obc, key_bdy key_agrif	non-linear not in reference not in reference
dynspg_tam.F90	dynspg_flt_tam.F90 dynspg_ts_tam.F90 dynspg_exp_tam.F90		not in reference not in reference
dynvor_tam.F90		dyn_vor_ens dyn_vor_een dyn_vor_ene dyn_vor_mix	not in reference not in reference
dynzdf_tam.F90	dynzdf_exp_tam.F90 dynzdf_imp_tam.F90		param. avmu and avmv as passive variables (if not: non-differentiability issue, see zdftke formulation) (should be treated as active var. when key_zdf_noncst is activated
dynmem_tam.F90			
wzvmod_tam.F90			

Table 1.3: Hand coding status of NEMOTAM, Dynamics part

Modules called by step_tam	Module(s) called by previous one	Options	Remarks
Ocean tracers (TRA)			
traadv_tam.F90			
	traadv_cen2_tam.F90		no upstream scheme (equivalent to zcofi=0)
	traadv_tvd_tam.F90		not differenciable
	traadv_muscl_tam.F90		not differentiable
	traadv_muscl2_tam.F90		not differentiable
	traadv_ubs_tam.F90		not differenciable
	traadv_qck_tam.F90		not differentiable
	traadv_eiv_tam.F90		not differentiable
trabbc_tam.F90			
tradmp_tam.F90			hlmp is saved in direct model to handle uslp, vslp
traldf_tam.F90			
		key_ldf_ano	uslp, vslp treated as passive
	traldf_bilap_tam.F90		not in reference
	traldf_bilapg_tam.F90		not in reference
	traldf_lap_tam.F90		pht_ldf, pst_ldf not include
tranxt_tam.F90			
		key_obc, key_bdy	not in reference
		key_agrif	not in reference
traqsr_tam.F90			
trasbc_tam.F90			
trazdf_tam.F90		lk_vvl	non-linear
	trazdf_imp_tam.F90	lk_vvl	wslpi, wslpj treated as passive
	trazdf_exp_tam.F90	lk_vvl	
Surface boundary conditions (SBC)			
sbcmod_tam.F90			
	sbc_gyre_tam.F90		
	sbc_flx_tam.F90		
	sbc_ssr_tam.F90		
	sbc_fwb_tam.F90		
	sbc_clo_tam.F90		
	sbc_ana_tam.F90		

Table 1.4: Hand coding status of NEMOTAM, Tracers and Surface Boundary Condition part

Used for	Functionalities	CPP keys	Associated namelist logical
Horizontal Diffusion/Viscosity			
	Lateral diffusion along isopycinal slopes	key_ldfslp	
Horizontal Tracer diffusion			
	Tracer lateral diffusion eddy induced velocity parametrization	key_traldf_eiv	
	Tracer lateral laplacian		ln_traldf_lap
	Iso-neutral application direction for operator		ln_traldf_iso
	Tracer lateral diffusion 1d=f(depth), 2d=f(lat,ion) or 3d=f(lat,ion, depth)	key_traldf_c2d	
Horizontal Dynamics Viscosity		nam_dynldf	
	Dynamic Laplacian operator lateral viscosity		ln_dynldf_lap
	Dynamic biLaplacian operator lateral viscosity		ln_dynldf_bilap
	Geopotential application direction viscosity		ln_dynldf_lap
	Dynamic lateral viscosity 1d, 2d or 3d	key_dynldf_3cd	
Vertical viscosity Diffusivity			
	Vertical diffusion based on TKE closure scheme	key_zdftke	namtke
	Vertical double diffusion mixing for salinity	key_zdfddm	namddm
Vertical convective processes			
	enhanced vertical diffusion parametrization		ln_zdfevd
Surface forcing fields			namsbc, key_forced_daily, key_tau_daily
Initial state and damping			
	Internal Newtonian tracers (T,S) damping	key_tradmp	namdtp
	Use a 3D monthly salinity data (Levitus)	key_dtasal	
	Use a monthly temperature data (Levitus)	key_dtatem	
	Use a monthly sst data (Levitus)	key_dtasst	
	Sea surface temperature damping		namsbc, namsbc_ssr, ln_ssr
	Sea surface salinity damping		namsbc, namsbc_ssr, ln_ssr

Table 1.5: Available option in NEMOTAM, part1

Used for	Functionalities	CPP keys	Associated namelist logical
On-line diagnostics			
	Eddy induced velocity (Gent-Williams parametrization)	key_diaeiv	
	Thermocline, 20°C-28°C isotherm depths	key_diahth	
	Enable observation operator	key_diaobs	
High Performances computing			
	Enable MPP computing with MPI	key_mpp_mpi	
		key_mpi_send	
	reproductibility	key_mpi_send	
Vertical coordinates			nam_zgr
	Full steps - Z		ln_zco
	Partial steps - PS		ln_zps
Pressure gradient treatment			
	Filtered free surface algorithm	key_dynspg_flt	
Vorticity scheme			nam_dynvor
	enstrophy conserving		ln_dynvor_ens
	energy and enstrophy conserving		ln_dynvor_een
Advection schemes			nam_traadv
	2nd order centered		ln_traadv_cen2
Configuration			
	Idealized Double Gyre configuration with a flat bottom	key_gyre	
	Global ORCA configuration at 2deg resolution	key_orca_r2	
	Open boundaries POMME configuration at 0.25° resolution	key_pomme_r025	
Other CPP keys			
	Geothermal boundary condition flux		nambbc
	Diffusive Bottom Boundary Layer	key_trabbl_dif	nambbl

Table 1.6: Available option in NEMOTAM, part2

1.7 The generic validation interface

1.7.1 Introduction

The validation interface aims at showing the consistency of the TAM with respect to the *direct* code. To fulfill this goal we demonstrate that:

1. the adjoint code is the adjoint of its tangent code
2. the tangent linear code is the tangent linear of the direct code

with defined criteria.

In general, TAM modules are subdivided into four parts:

1. a tangent-linear routine
2. an adjoint routine
3. an adjoint test routine
4. a tangent-linear routine

Item 3 and 4 are optional. The strategy is to perform adjoint and tangent-linear tests for all critical sub-modules and the main module *step_tam* for time-step integration.

1.7.2 Adjoint test

By definition, A^* (or ${}^t\mathrm{A}$ in a finite-dimension case) is the adjoint operator of a linear operator A defined on a Hilbert space if it has the following property:

$$(Ax, y) = (x, {}^tAy) \tag{1.4}$$

In practice, we apply the comparison to random perturbations δx in the model-space and δy^* in the adjoint model-space. We have then:

$$(A\delta x, \delta y^*) = (\delta x, {}^tA\delta y^*) \tag{1.5}$$

Adjoint test requirements

- A saved trajectory of the direct model
- An additionnal namelist (*namtst* for adjoint test, *namtst_tlm* for tangent test

Unitary test versus Global test

Unitary test involves one or a limited set of routines. The pertubation vectors δX and δY are the input/output variables identified as actives variables. There is no time evolution. Global test involves time evolution (loop on *step_tam*) and δX and δY are input/output variables identified as the *control vector* (or variables of main interest).

Test Pass Criteria

We note (if exists) the relative error Err. The pass test criteria is based on the comparison of Err with the epsilon machine $\varepsilon_{machine} = eps$.

$$Err = \frac{(A\delta x, \delta y^*) - (\delta x, {}^tA\delta y^*)}{(A\delta x, \delta y^*)} \tag{1.6}$$

The adjoint test results output is the file *adjoint_test.output_xxx* with the following label:

Test status is OK	$Err \leq Eps$
test status is WARNING	$Eps < Err \leq 100 \times Eps$
Test status is FAILED	$Err > 100 \times Eps$

Table 1.7: Adjoint test labels definition

Run the adjoint test

- set the directory where the direct trajectory is stored
- in the namelist *namtst*
  - set *ln_tst_nemotam* to *.true.* (switch for M adjoint and tangent tests)
  - set *ln_tst_cpd_tam* to *.true.* (switch for adjoint tests of the sub-components)
  - set *ln_tst_stp_tam* to *.true.* (switch for adjoint tests of main components *step_tam*)
  - set *ln_tst_tan_cpd* and *ln_tst_tan* to *.false.* (switch for tangent tests)

1.7.3 Tangent-Linear validation

The tangent validation test checks that the Tangent Linear model L is a first-order approximation of the direct model $\mathcal{M}$. Then, we have the following second-order Taylor expansion:

$$\mathcal{M}(X + p\delta X) = \mathcal{M}(X) + pL(\delta X) + \varepsilon \tag{1.7}$$

When the factor p tends to zero:

1. $\mathrm{N}_p = \mathcal{M}(X + p\delta X) - \mathcal{M}(X)$ tends to $\mathrm{L}_p = pL(\delta X)$, first-order validation
2. ε behaves as $\mathcal{O}(p^2)$, second-order validation

this means that for a given p_0, for p smaller than p_0 the error of Tangent Linear with respect to the model is decreasing as p^2.

Note that the second test (b) is not relevant if $\mathcal{M}$ is linear.

Test	Expression	behaviour
a	$\frac{Norm(\mathrm{N}_p,.)}{Norm(\mathrm{L}_p,.)}$	1
b	$\frac{Norm(\mathrm{N}_p-\mathrm{L}_p,.)}{p^2}$	Cst

Table 1.8: Tangent Linear tests. First-order check (a) and second-order check (b)

Build the tangent test

The tangent test is embedded into nemotam. To build it the user must add the compilation key *key_tst_tlm* in the CPP key set. Otherwise, a error message will raise during the run.

Run the tangent test

To run the tangent test the code must

- set the directory where the direct trajectory is stored
- in the namelist *namtst*
  - set *ln_tst* to *.true.*
  - set *ln_tst_nemotam* to *.true.* (switch for M adjoint and tangent tests)
  - set *ln_tst_tan_cpd* or *ln_tst_tan* to *.true.* (switch for tangent tests)
  - set *ln_tst_cpd_tam* and *ln_tst_stp_tam* to *.false.* (switch for adjoint tests)
- in the namelist *namtst_tlm*
  - set *tlm_bch* to *.true.* (branching: '0' for $\mathcal{M}(X)$, '1' for $\mathcal{M}(X+p\delta X)$ and '2' for L_{pdX})
  - set *curr_loop* to *.true.* (current iteration)
  - set *h_ratio* to *.true.* (value of the ratio factor p applied to δX)

```
!-----------------------------------------------------------------------
!       namtst_tlm    tangent test parameters
!-----------------------------------------------------------------------
!
! cur_loop = current loop iteration
! h_ratio  = current h_ratio
! tlm_bch  = flag for branching
!          = 0  :M(X)           , direct without perturbation
!          = 1  :M(X+h_ratio.dX), direct with perturbation
!          = 2  :L(h_ratio.dX)  , linear
&namtst_tlm
  tlm_bch  = 1
  cur_loop = 0
  h_ratio  = 10
 /
```

Tangent test ouputs

The output is saved with the name *tan_diag.output_0000*. The header give information on the computed parameters. For iteration n, the output gives the follwing data:
routine name, $N_n = \mathcal{M}(X+p_n\delta X)-\mathcal{M}(X)$, $E_n = N_n/\mathrm{L}(p_n\delta X)$, $Er_n = (N_n-\mathrm{L}(p_n\delta X))/\mathrm{L}(p_n\delta X)$, $\mathrm{L}(p_n\delta X)$, $N_n - \mathrm{L}(p_n\delta X)$, $(E_n - 1)/p_n$, Er_n/p_n.
To have an overview of the behaviour of the tangent iteration can be made by the user changing the factor ratio p_n. The factor takes usually a geometric progression as: $p_n = p_0 r^{-n}$, with initial value p_0 and common ratio r.

1.7.4 Tangent-Linear Hypothesis (HLT) validity

The innovation vector validity relies on the validity of the Tangent Linear. Before any 4DVar assimilation we must insure that the tangent model is valid within our assimilation time-window. Otherwise the optimisation may fail as the cost function gradient will be degraded or even wrong. We define a maximum assimilation time-window T_a^{max}. This time-window will be different for each configuration. According to each application the user its own criteria to define to define T_a^{max} to fulfill its needs. The size of T_a^{max} may vary from 1 to 3 months (**???**).
To estimate T_a^{max} we study the behaviour of $y_1 = \mathcal{M}_{t_0\to t}(x_0+\delta x)$ with respect to $y_2 = \mathcal{M}_{t_0\to t}(x_0)+\mathrm{L}_{t_0\to t}(\delta x)$.
The Tangent Linear will be valid as long as the difference $y_1 - y_2$ is "small". As discussed in (**?**) definition criteria of T_a^{max} will rely on threshold on the *RMS* of $y_1 - y_2$ and the coefficient correlation $\rho_{y_1y_2}$. A correlation coefficient $\rho_{y_1y_2}$ above 90% is considered as very good.

$$RMS(y_1 - y_2) = \sqrt{\frac{\sum_i^n (y_1^i - y_2^i)^2}{n}} \quad \text{, n: number of ocean points} \tag{1.8}$$

$$\rho_{y_1y_2} = \frac{\sum_i^n (y_1^i - \bar{y_1}) \cdot (y_2^i - \bar{y_1})}{\sqrt{\sum_i^n (y_1^i - \bar{y_1})^2} \cdot \sqrt{\sum_i^n (y_2^i - \bar{y_2})^2}} \quad \text{, n: number of ocean points} \tag{1.9}$$

The initial condition as well as the increment δx are critical to define T_a^{max}. The user should work with a typical increment δx of its configuration.

HLT test tool

NEMOTAM offers the possibility to analize the Tangent-Linear Hypothesis through a dedicate executable *model_hlt.exe*. The increments δx is either given by the user either computed from two restarts file x_0 and x_1 (then $\delta x = x_0 - x_1$). To compute y_1 and y_2, *model_hlt.exe* should be run three times: two for the direct model trajectories ($\mathcal{M}_{t_0\to t}(x_0 + \delta x)$ and $\mathcal{M}_{t_0\to t}(x_0)$) and one for

the tangent trajectory ($L_{t_0 \to t}(\delta x)$). It is driven through the namelists *namhlt* and *tl_tamtrj*.

```
!-----------------------------------------------------------------------
!       namhlt    tangent-linear hypothesis test parameters
!  nstg    flag for to compute model with or without perturbation
!          0: M(X)
!          1: M(X+dX)
!          2: L(dX)
!  ln_hltt    Logical switch for applying temperature increments
!  ln_hlts    Logical switch for applying salinity increments
!  ln_hltuv   Logical switch for applying horizontal velocity increments
!  ln_hltssh  Logical switch for applying sea surface height increments
!  ln_incdx   Logical swith to compute dx (F) or reading from file (T)
!  ln_hnorm   Logical swith to normalization (T) or not (F) using rhstdX
!  rhstdt     Upper bound of normalization factor for temperature
!  rhstds     Upper bound of normalization factor for salinity
!  rhstduv    Upper bound of normalization factor for velocity
!  rhstdssh   Upper bound of normalization factor for sea surface height
!-----------------------------------------------------------------------
&namhlt
    nstg = 2
    ln_hltt   = .TRUE.
    ln_hlts   = .TRUE.
    ln_hltuv  = .TRUE.
    ln_hltssh = .FALSE.
    c_hltinc = ""
    ln_incdx = .FALSE.
    ln_hnorm = .TRUE.
    rhstdt   = 1.
    rhstds   = 0.1
    rhstduv  = 0.01
    rhstdssh = 0.01
/
!-----------------------------------------------------------------------
!       namtl_trj    tangent-linear trajectory frequency output
!  ln_trjwri_tan Logical switch rajectory output
!  nittrjfrq_tan frequency output for linear tangent
!-----------------------------------------------------------------------
&namtl_trj
    ln_trjwri_tan = .TRUE.
    nittrjfrq_tan = 96
/
```

1.8 MPI parallelization

1.8.1 General features

We remind some basics for coding the corresponding adjoint of the tangent linear part:

Operation	Tangent-Linear	Adjoint
communication	send	receive
communication	receive	send
communication	gather	scatter
arithmetic	scatter	gather
inpout/Output	write	read
inpout/Output	read	write

Table 1.9: ORCA2 Results of Adjoint test fo sub-routines

There is no tangent of *lbc_lnk*. The direct routine is used. For the adjoint part we use the coding rule described in table 1.9

1.8.2 Adjoint of MPI_ALLGATHER

From revision 3.2 of NEMO a new treatment of north fold horizontal bondary condition is performed using lbcnfd.F90 routine. This induces a MPI_ALLGATHER usage in libmpp.F90 module (mpp_lbc_north_3d and mpp_lbc_north_2d).
MPI_ALLGATHER gathers data from all tasks and distribute it to all.

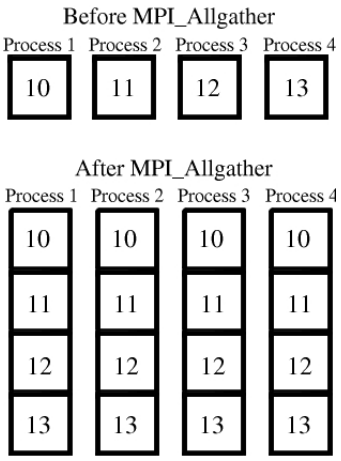


Figure 1.1: MPI_ALLGATHER task description (ref: `http://rc.usf.edu/tutorials/classes/tutorial/mpi/chapter8.html`

If we call $zloc_P_i$ the data from task (i) and the $zglo$ the result of MPI_ALLGATHER and $zloc_adj_P_i$, $zglo_adj$ their adjoint equivalent.
The adjoint counter-part of

```
MPI_ALLGATHER(zglo, zloc)
```

is

```
MPI_SUM(zglo_adj)
zloc__Pi = zglo_adj(:,i)
```

As short demonstration, we consider a two-task case P_1 and P_2.
MPI_ALLGATHER(zglo, zloc) is equivalent to:

for P_1:

```
(1) send and receive local data:
send(zloc1_P1, P1, P2)    --->     zloc1_P2 = zloc1_P1
recv(zloc2_P1, P2, P1)    --->     zloc2_P1 = zloc2_P2

( zlocK_Pj : local data K of j task)

B- zglo operation
zglo_P1(:,1) = zloc1_P1
zglo_P1(:,2) = zloc2_P1
```

we then write the adjoint as follows:

```
zloc2_P1*     = zloc2_P1* + zglo_P1*(:,2)
zglo_P1*(:,2) = 0
zloc1_P1*     = zloc1_P1* + zglo_P1*(:,1)
zglo_P1*(:,1) = 0

zglo_tot*     = zglo_P1* + zglo_P2*   (i.e mpp_sum_nfd)

(! because zloc_P1* is not null here, thus: zloc1_P1* = zloc_P1* + zloc1_P2*)

zloc1_P1*     = zglo_tot*(:,1)
zloc2_P1*     = zglo_tot*(:,2)
```

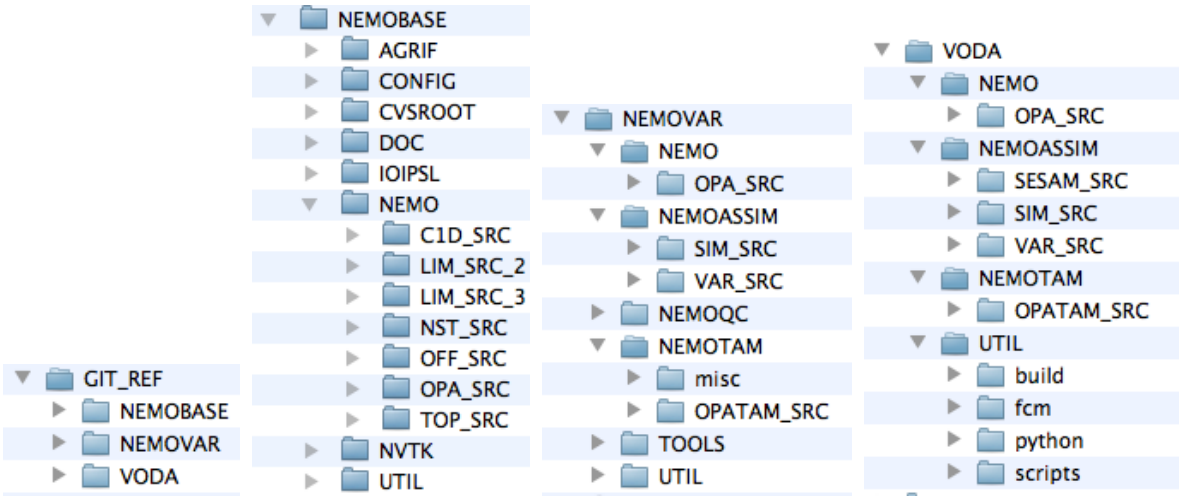
Chapter 2

NEMOTAM USER'S GUIDE

2.1 Source code structuration

The project will work with three repositories (see Fig. 2.1b,c,d):

- a NEMOBASE repository, with an official NEMO released version currently the tag nemo_v3, available at :
  https://forge.ipsl.jussieu.fr/nemo/browser/tags/nemo_v3
- a NEMOVAR repository, with all developments related to the nemovar project, including new and modified files for NEMO, this is the standard version of NEMOVAR (currently tag N3.0-A2.0)
- a VODA repository including new development for NEMOTAM and NEMOVAR within the VODA project



(a) GIT tree (b) NEMOBASE (c) NEMOVAR (d) VODA

Figure 2.1: Code organization

The building process will look for both repositories, if any redundancy is found, a VODA files will overwrite the corresponding NEMOVAR file, and a NEMOVAR file will overwrite the corresponding NEMOBASE file in the building folder.

The NEMOBASE repository (see Fig. 2.1b) is the standard NEMO folder. The NEMOVAR repository (see Fig. 2.1c) is subdivided into 6 components: NEMO, NEMOQC, NEMOTAM, NEMOASSIM, UTIL and TOOL. The NEMOTAM folder is a mimic of the NEMO folder (meaning we have an OPATAM_SRC folder subdivided into components as ASM, DOM, DYN, LDF, TRA, ...). The VODA folder (see Fig. 2.1d) mimic the NEMOVAR folder, with additional contributions. This organization gives a clear snapshot of what is being developped and lets some room for further development.

2.2 Getting started tutorial

2.2.1 Installing the code using the NEMOVAR GIT repository

Checking out the code from the GIT repository

You need to ask the VODA project team to have developer access to the VODA project on the gforge.inria.fr server and access to the NEMO - IPSL. You can then follow the steps described below.

```
Create first a new directory, where the source code will be installed.
For instance,
cd $VODA_GIT  (=$HOME/VODA-PROJECT/VODA_GIT e.g.)

---> Install NEMOBASE
go to https://forge.ipsl.jussieu.fr/nemo/browser/tags/nemo\_v3 (authentication
    required)
and click on 'zip archive' at the bottom of the page.
put this archive in $VODA_GIT, unzip it and rename the obtained directory
   NEMOBASE (this name is mandatory)

---> Install NEMOVAR
Download the following bundles, located on the gforge server under the "files"
    tab:
NEMOVAR: N3.0-A2.0 => nemovar_N3.0-A2.0.bundle
and put them in the $VODA_GIT/BUNDLES directory.

In $VODA_GIT, you now have:
tree BUNDLES/
BUNDLES/
|-- NEMOBASE.bundle.20090902
`-- nemovar_N3.0-A2.0.bundle

Clone these bundles :
git clone BUNDLES/NEMOBASE.bundle.20090902 NEMOBASE/
git clone BUNDLES/nemovar_N3.0-A2.0.bundle NEMOVAR/
```

```
---> Install VODA
git clone git+ssh://cdeltel@scm.gforge.inria.fr//gitroot/voda/voda.git VODA/

--> Final structure of the $VODA_GIT directory

tree -d -L 2 $VODA_GIT

VODA_GIT
|-- BUNDLES
|-- NEMOBASE
|   |-- AGRIF
|   |-- CONFIG
|   |-- CVSROOT
|   |-- DOC
|   |-- IOIPSL
|   |-- NEMO
|   |-- NVTK
|   `-- UTIL
|-- NEMOVAR
|   |-- NEMO
|   |-- NEMOASSIM
|   |-- NEMOQC
|   |-- NEMOTAM
|   |-- TOOLS
|   `-- UTIL
`-- VODA
    |-- NEMO
    |-- NEMOASSIM
    |-- NEMOTAM
    `-- UTIL
```

Building the code (in default mode : ORCA2 validation interface)

NEMOTAM makes use of the fcm based NEMOVAR compiling environment, a simple call to VODA/UTIL/build/fcmvmake.ksh with the usual arguments and the `nemoall` keyword should do the trick.

```
> fcmvmake.ksh -c $COMPILE -t $WORKDIR -B nemoall
```

the above requires that you create a `VODA/UTIL/build/fcmconfig/bld/$COMPILE/nemo.cfg` describing your compilation environment (examples are given in the same directory).

How to run the code is described in section 2.3.

2.2.2 Installing the code using the NEMOTAM SVN repository

Checking out the code from the SVN repository

NEMOTAM can be extracted using svn like NEMO. People should first to register on NEMO website www.nemo-ocean.eu. Then NEMOTAM is available through modipsl framework, so modipsl has to be extracted first.

• Extract modipsl:

```
svn co http://forge.ipsl.jussieu.fr/igcmg/svn/modipsl/trunk modipsl
```

• Extract NEMOTAM:

```
cd modipsl/util
./model NEMOTAM
```

Building the code (in default mode : ORCA2 validation interface)

Choose the configuration to build. You can edit the script fait_config_tam to add a new one.

```
cd modipsl/modeles/UTIL
./fait_config_tam ORCA2_TAM
```

You can build the direct model in the same way:

```
cd modipsl/modeles/UTIL
./fait_config ORCA2_LIM
```

To create the Makefiles, edit first modipsl/util/AA_make.gdef to find or add your compiler, then:

```
cd modipsl/util
./ins_make
```

To compile:

```
cd modipsl/modeles/NEMOTAM/WORK
gmake
```

2.2.3 Which executable for which test?

To run a bunch of tests to verify your configuration you need to compile several executable according to your test target. The graph 2.2 give an overview of the relationship between the executable and the available tests.

Running the code

2.3 Demonstrators

2.3.1 The ORCA2 global configuration

ORCA is the generic name given to global ocean configurations. Its specificity lies on the horizontal curvilinear mesh used to overcome the North Pole singularity found for geographical meshes. ORCA is based on a 2 degrees Mercator mesh, (i.e variation of meridian scale factor as cosinus of the latitude). In the northern hemisphere the mesh has two poles so that the ratio of anisotropy is nearly one everywhere. The vertical domain spreads from the surface to a depth of 5000m. There are 31 levels, with 10 levels in the to 100m. The time step depends on the resolution. It is $1\text{h}36'$ for ORCA2 so that there is 15 time steps in one day.

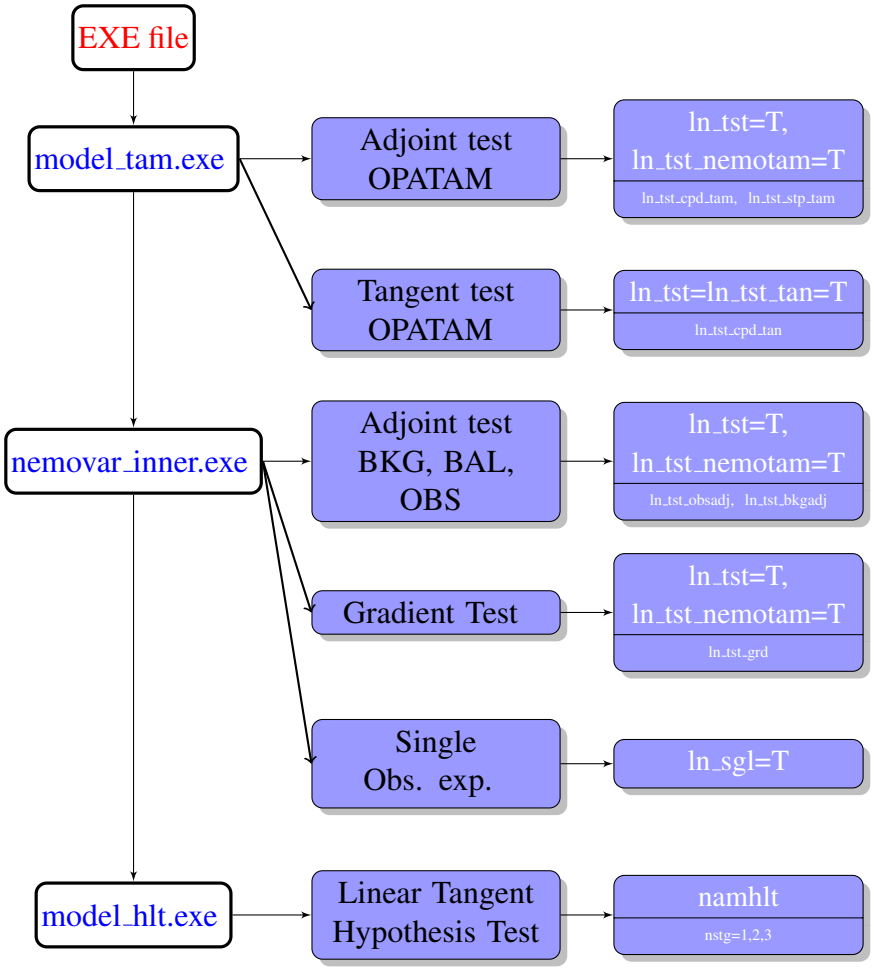


Figure 2.2: Releationships between executable files (first column) and available tests (second column). The thrid column focuses on the main namelist's parameters ('*namtst*' for adjoint, gradient, tangent tests, '*namobs*' for single obs.experiement and '*namhlt*' for linear tangent hypothesis)

How to compile the ORCA2 configuration ?

You must first follow the steps described in 2.2.1. The compilation process described below then relies upon the FCM (Flexible Configuration Manager) toolbox, which is included in VODA tree, see e.g. the following directory :

```
ls $VODA_GIT/VODA/UTIL/fcm/bin
```

Then, you may proceed in the following manner:

- Get sample input data, needed for the model to run (`http://ljk.imag.fr/membres/Arthur.Vidard/docs/nemov3_testdata.tar.gz`) :
- Customize the first configuration file **nemo.cfg**, which is located according to your platform architecture (mac_intel here) :

  ```
  vi $VODA_GIT/VODA/UTIL/build/fcmconfig/bld/mac_intel/nemo.cfg
  ```

- Customize the second configuration file, with CPP keys: **cppkeys.ORCA2_Z31.cfg**. A sample file is ready to be used, which you can modify:

  ```
  vi $VODA_GIT/VODA/UTIL/build/fcmconfig/nemo/cppkeys.ORCA2_Z31.cfg
  ```

- You are now ready to compile the code. You need to create a "scratch" directory (for example SCRATCH_ORCA2 below), where compilation and execution will be performed:

  ```
  cd $VODA_GIT/VODA/UTIL/build
  ./fcmvmake.ksh -t $SCRATCH_ORCA2 -c mac_intel -B NEMO    -G ORCA2_Z31
  ./fcmvmake.ksh -t $SCRATCH_ORCA2 -c mac_intel -B NEMOTAM -G ORCA2_Z31
  ```

How to run the ORCA2 configuration ?

A sample minimal script is given, just as an example : The *direct* model NEMO

```
$VODA_GIT/VODA/UTIL/scripts/runnemo.ksh  -t $SCRATCH_ORCA2 -A mac_intel -v
   NEMO        -g ORCA2_Z31
```

The *adjoint* model NEMO tests

```
$VODA_GIT/VODA/UTIL/scripts/runnemo.ksh  -t $SCRATCH_ORCA2 -A mac_intel -j $
   TRAJ       -v NEMOTAM -g ORCA2_Z31
```

where `$TRAJ` is the reference trajectory directory. This script absolutely needs to be customized with respect to your working environment. Please have a look at this script before proceeding further. You will first need to create a reference trajectory by running only the *direct* model NEMO, and you then may want to use NEMOTAM to

- launch the generic TAM test interface,
- investigate the time span over which the tangent linear hypothethis remains valid,
- etc.

Testing the tangent model and the tangent linear hypothesis

As discussed previously, the aims of the test is to illustrate the correctness of the tangent and how we have dealt with the non-linearity (if it exits) of the direct model.

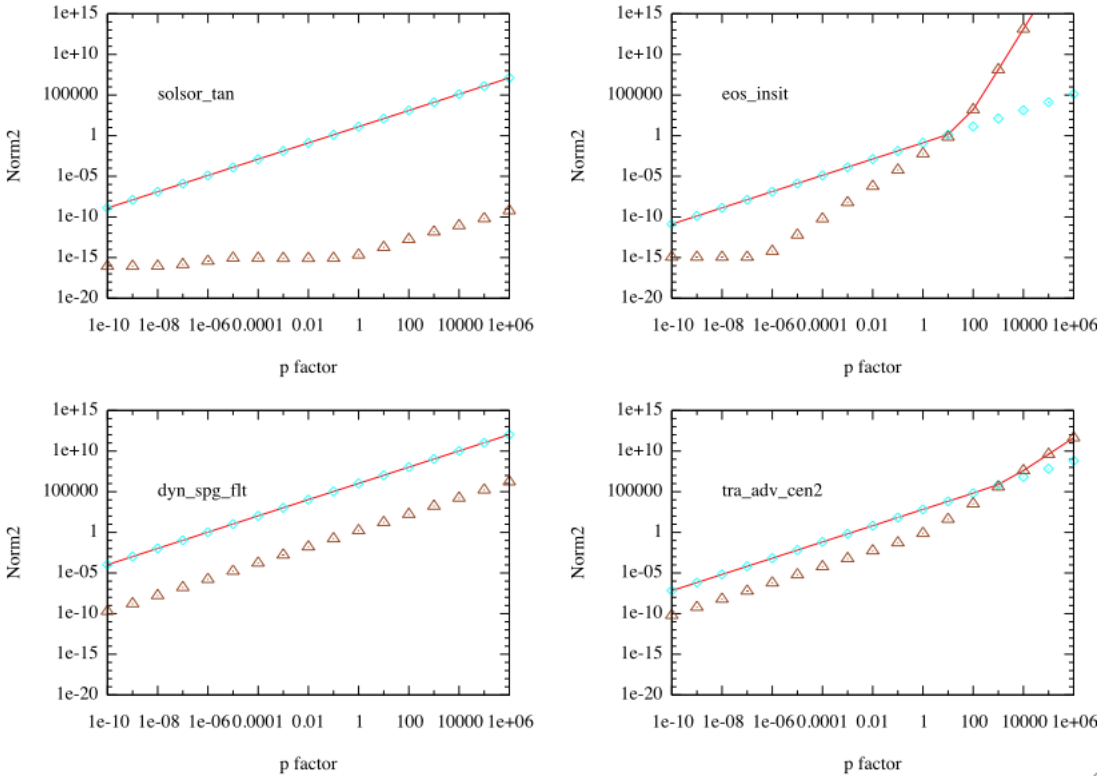


Figure 2.3: Tangent test type: Four kinds of test results. Top left (a), linear routine with a residual following the error machine. Top right (b), non-linear routine with a residual decreasing as p-order of 2. Bottom left (c), linear routine with a residual decreasing as p-order of 1. Bottom right (d), non-linear routine with a residual decreasing as p-order of 1. (Legend: Solid line is $N_p = M(x + p\delta x) - M(x)$, circle symbol $L_p = L(p\delta x)$, triangle symbol $N_p - L_p$)

Discussion

We have divided the results into four categories (a), (b), (c) and (d) for analysis (see table 2.1). Category (a), illustrated by *solsor_tan* in figure 2.3, gathers tangent-linear routines behaving as expected w.r.t to their direct routine (also linear) counter-part: the computed residual is close to the error machine (the difference $\frac{L_p}{N_p - L_p}$ is around 10^{-15}. Category (b), illustrated by *eos_insitu_tan* in figure 2.3, gathers tangent-linear routines behaving as expected w.r.t to their direct routine (not linear) counter-part: the computed residual is equivalent to a $\mathcal{O}(p^2)$. Categories (c) and (d) , illustrated by *dynspg_flt_tan* and *tra_adv_cen2_tan* in figure 2.3, gather suspicious cases we need to validate (or invalidate) through the choice made for the tangent-linear. Case (c) is for direct linear routine and (d) direct non-linear routine.

Routine (L)	Category class
solsor_tan	a
dynadv_tan	b
dynhpg_tan	a
traldf_lap_tan	b
trazdf_imp_tan	b or c ?
tra_zpshde_tan	b
trasbc_tan	b
bn2_tan	b
eos_1pt_tan	b
eos_insitu2d_tan	b
eos_insp_tan	b
eos_insitu_tan	b
dynspg_flt_tan	c
traadv_cen2_flt_tan	d

Table 2.1: ORCA2 Results of Tangent tests for sub-routines

The case of *traadv_cen2_flt_tan* is fully understood as an approximation was done on the numerical scheme (the tangent linear does not include the upstream component of the numerical scheme). Below a certain threshold the upstream part (which is a $\mathcal{O}(p)$ in N_p) is no longer small compared to the second order scheme and thus, L_p.

Testing the adjoint model

The adjoint test for the ORCA2 configuration has proven satisfactory for the unitary routine tests (see table 2.2) and the global test (see tables 2.3 and 2.4. For the latter, the integration time tested varied from 1 to 15 days.

Routine (L)	$(L\delta x)^\intercal \cdot W dy$	$\delta dx^\intercal \cdot L^\intercal W \delta y$	Rel. Err.	Mach. eps.	Status
sol_sor_adj	.525406844734452E+15	.525406844734452E+15	.0E+00	.2E-14	ok
sbcfwb_adj 1 1	.711617836790924E+12	.711617836790924E+12	.7E-15	.2E-14	ok
sbcfwb_adj 2 1	.711586863123878E+12	.711586863123878E+12	.0E+00	.2E-14	ok
sbcfwb_adj 1 2	.721405227628168E+12	.721405227628168E+12	.0E+00	.2E-14	ok
sbcfwb_adj 2 2	.721331456200554E+12	.721331456200554E+12	.0E+00	.2E-14	ok
sbcfwb_adj 1 3	.711617836790924E+12	.711617836790924E+12	.7E-15	.2E-14	ok
sbcfwb_adj 2 3	.711586863123878E+12	.711586863123878E+12	.0E+00	.2E-14	ok
sbc_ssr_adj	.289793337319649E+21	.289793337319649E+21	.2E-15	.2E-14	ok
sbc_gyre_adj 1	.575805928604001E+19	.575805928604001E+19	.0E+00	.2E-14	ok
sbc_gyre_adj 2	.575805928604001E+19	.575805928604001E+19	.0E+00	.2E-14	ok
sbc_ssm_adj	.585782123944269E+15	.585782123944269E+15	.0E+00	.2E-14	ok
bn2_adj	.763641537078732E+10	.763641537078733E+10	.5E-15	.2E-14	ok
div_cla_adj	.134110702040363E+17	.134110702040363E+17	.1E-15	.2E-14	ok
div_cur_adj T1	.307633707985862E+07	.307633707985862E+07	.0E+00	.2E-14	ok
div_cur_adj T2	.321541201767843E+07	.321541201767843E+07	.6E-15	.2E-14	ok
dyn_adv_adj	.252818496373574E+17	.252818496373574E+17	.2E-15	.2E-14	ok
dynvor_adj ens	.252818508381207E+17	.252818508381207E+17	.0E+00	.2E-14	ok
dynvor_adj een	.252818506922870E+17	.252818506922870E+17	.0E+00	.2E-14	ok
dynldf_adj lap	.311731217229964E+17	.311731217229965E+17	.3E-15	.2E-14	ok
dynldf_adj blp	.252575821916828E+17	.252575821916828E+17	.2E-15	.2E-14	ok
dyn_zdf_adj	.252556994319522E+17	.252556994319522E+17	.0E+00	.2E-14	ok
dyn_spg_flt T1	.790035995408866E+24	.790035995408866E+24	.0E+00	.2E-14	ok
dyn_spg_flt T2	.315978990829171E+25	.315978990829171E+25	.3E-15	.2E-14	ok
dyn_hpg_adj	.253953510430215E+17	.253953510430214E+17	.3E-15	.2E-14	ok
dyn_nxt_adj	.758385105599429E+17	.758385105599429E+17	.0E+00	.2E-14	ok
wzv_adj	.258438017479678E+13	.258438017479678E+13	.0E+00	.2E-14	ok
tra_sbc_adj	.517847966550809E+15	.517847966550810E+15	.5E-15	.2E-14	ok
tra_qsr_adj	1 .134738569058199E+19	.134738569058200E+19	.1E-14	.2E-14	ok
tra_qsr_adj	2 .134738569058199E+19	.134738569058200E+19	.1E-14	.2E-14	ok
tra_dmp_adj T1	.152739703065678E+19	.152739703065678E+19	.0E+00	.2E-14	ok
tra_dmp_adj T2	.152739703065678E+19	.152739703065678E+19	.0E+00	.2E-14	ok
tra_dmp_adj T3	.152739703065678E+19	.152739703065678E+19	.0E+00	.2E-14	ok
tra_adv_cen2	.152558770733990E+19	.152558770733990E+19	.1E-14	.2E-14	ok
tra_cla_adj Gi	.487513953864278E+16	.487513953864278E+16	.2E-15	.2E-14	ok
tra_cla_adj BM	.158030397951644E+15	.158030397951644E+15	.6E-15	.2E-14	ok
tra_ldf_iso_ad	.152906929763072E+19	.152906929763072E+19	.0E+00	.2E-14	ok
trazdf_exp_adj	.235296147886893E+37	.235296147886893E+37	.6E-15	.2E-14	ok
trazdfimpadjT1	.208237542411001E+25	.208237542411001E+25	.5E-15	.2E-14	ok
trazdfimpadjT2	.208237542411001E+25	.208237542411001E+25	.5E-15	.2E-14	ok
trazdfimpadjT3	.208237542411001E+25	.208237542411001E+25	.5E-15	.2E-14	ok
tra_nxt_adj	.405901725366058E+19	.405901725366058E+19	.9E-15	.2E-14	ok
eos_adj insitu	.405930021470975E+11	.405930021470975E+11	.0E+00	.2E-14	ok
eos_adj pot	.241698506116296E+17	.241698506116297E+17	.5E-15	.2E-14	ok
eos_adj 2d	.249470161251995E+09	.249470161251995E+09	.4E-15	.2E-14	ok
eos_adj 1pt	.108202320860649E+10	.108202320860649E+10	.2E-15	.2E-14	ok
zps_hde_adj	.417898655654515E+18	.417898655654515E+18	.2E-15	.2E-14	ok
istate_tst	.140086193684597E+19	.140086193684597E+19	.7E-15	.2E-14	ok

Table 2.2: ORCA2 Results of the Adjoint tests for the main sub-routines

Routine (L)	$(L\delta x)^\intercal \cdot W dy$	$\delta dx^\intercal \cdot L^\intercal W \delta y$	Rel. Err.	Mach. eps.	Status
step_adj U	.161275281471349E+18	.161275281471349E+18	.6E-15	.2E-14	ok
step_adj V	.318648865958325E+18	.318648865958325E+18	.4E-15	.2E-14	ok
step_adj T	.132120109179127E+19	.132120109179127E+19	.0E+00	.2E-14	ok
step_adj S	.281362832449564E+18	.281362832449564E+18	.0E+00	.2E-14	ok
step_adj SSH	.509250348807475E+10	.509250348807474E+10	.2E-14	.2E-14	ok
step_adj	.206971044341451E+19	.206971044341451E+19	.1E-15	.2E-14	ok

Table 2.3: ORCA2 Results of the Adjoint tests for the main routine *step_tam* for a run-window of one day. From line one to four, each control parameter is perturbed individually. Column six, all control parameter are perturbed.

Routine (L)	$(L\delta x)^\intercal \cdot W dy$	$\delta dx^\intercal \cdot L^\intercal W \delta y$	Rel. Err.	Mach. eps.	Status
step_adj U	.102643067009108E+18	.102643067009108E+18	.2E-15	.2E-14	ok
step_adj V	.167827302823834E+18	.167827302823834E+18	.4E-15	.2E-14	ok
step_adj T	.893406215846414E+18	.893406215846413E+18	.7E-15	.2E-14	ok
step_adj S	.220114453479612E+18	.220114453479612E+18	.4E-15	.2E-14	ok
step_adj SSH	.265003707314821E+10	.265003707314821E+10	.4E-15	.2E-14	ok
step_adj	.137645166426475E+19	.137645166426475E+19	.6E-15	.2E-14	ok

Table 2.4: ORCA2 Results of the Adjoint tests for the main routine *step_tam* for a run-window of five days. From line one to four, each control parameter is perturbed individually. Column six, all control parameter are perturbed.

2.3.2 The GYRE regional configuration

GYRE is an idealized configuration representing double gyres in the Northern hemisphere. It is β-plane with a regular grid spacing at 1 degree horizontal resolution. There is 31 verticles levels. The configuration is forced with analytical heat, freswater and wind-stress fields.

How to compile the GYRE configuration ?

Similar to ORCA2 configuration replacing ORCA2_Z31 by GYRE_Z31.

How to run the GYRE configuration ?

Similar to ORCA2 configuration replacing ORCA2_Z31 by GYRE_Z31.

Tangent-Linear Hypothesis (HLT) validity

To illustrate the tangent linear hypothesis validity test We use an initial state x_0 ten years from spin-up and an increment δx computed from an previous assimiilation with ($|\delta x_T| \leq 1$, $|\delta x_S| \leq 0.1$, $|\delta x_U| \leq 0.0022$ and $|\delta x_V| \leq 0.0019$) . see figure 2.4

here: place more stats on δx (mean, min, max, std) ?

here: add comments for the graph

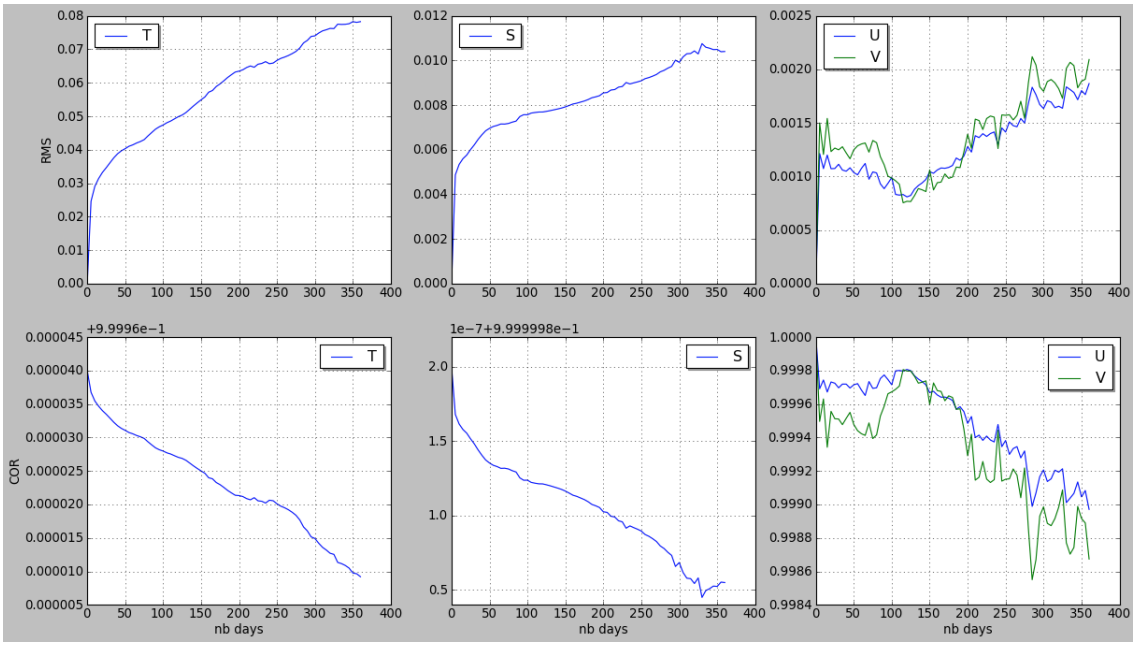


Figure 2.4: **Tangent linear hypothsesis validty for *GYRE_Z31***: plot in function of time-window in days over 1 year. Top graphs are RMS of T, S, U and V variables in S.I units. Bottom graphs are related coefficient correlation.

2.3.3 The POMME configuration (toy model for handling open boundaries)

Description of the configuration

NEMO is distributed with some reference configurations allowing the users to set up a first application, and the developers to validate their developments. POMME is a small square box configuration ($30 \times 40 \times 46$ grid points), extracted from NEMO DRAKKAR / ORCA025. All boundaries are open ocean boundaries (including the corners). This configuration was set up to facilitate developments and tests for regional configurations (however, it is not yet an official NEMO reference configuration). It should be soon included in the NVTK validation toolkit.

How to compile the POMME configuration ?

You must first follow the steps described in 2.2.1. The compilation process described below then relies upon the FCM (Flexible Configuration Manager) toolbox, which is included in VODA tree, see e.g. the following directory :

```
ls $VODA_GIT/VODA/UTIL/fcm/bin
```

Then, you may proceed in the following manner:

- Get sample input data, needed for the model to run:

  ```
  wget http://www.locean-ipsl.upmc.fr/~cdlod/data/data_POMME_Z46.tar.gz
  ```

- Customize the first configuration file **nemo.cfg**, which is located according to your platform architecture (mac_intel here) :

```
vi $VODA_GIT/VODA/UTIL/build/fcmconfig/bld/mac_intel/nemo.cfg
```

- Customize the second configuration file, with CPP keys: **cppkeys.POMME_Z46.cfg**. A sample file is ready to be used, which you can modify:

```
vi $VODA_GIT/VODA/UTIL/build/fcmconfig/nemo/cppkeys.POMME_Z46.cfg
```

- You are now ready to compile the code. You need to create a "scratch" directory (for example SCRATCH_POMME below), where compilation and execution will be performed:

```
cd $VODA_GIT/VODA/UTIL/build
./fcmvmake.ksh -t $SCRATCH_POMME -c mac_intel -B NEMO    -G POMME_Z46
./fcmvmake.ksh -t $SCRATCH_POMME -c mac_intel -B NEMOTAM -G POMME_Z46
```

How to run the POMME configuration ?

A sample minimal script is given, just as an example :

```
vi $VODA_GIT/VODA/UTIL/scripts/run_pomme.ksh
```

This script absolutely needs to be customized with respect to your working environment. Please have a look at this script before proceeding further. You will first need to create a reference trajectory by running only the *direct* model NEMO, and you then may want to use NEMOTAM to

- launch the generic TAM test interface,
- investigate the time span over which the tangent linear hypothethis remains valid,
- etc.

Testing the tangent model and the tangent linear hypothesis

coming soon

Testing the adjoint model

coming soon

2.4 How-to update NEMOTAM with your contribution?

2.4.1 Coding rules for TAM

NEMOTAM modules should follow the NEMO convention. However additional rules has to be followed:

- For a given direct module *mod.F90 (that contains the subroutine sub), a corresponding module mod_tam.F90 has to be created and it should contain the tangent linear subroutine called sub_tan, the adjoint subroutine sub_adj and the corresponding testing routine sub_adj_tst.*

- *For a given active direct variable var, the corresponding tangent linear and adjoint variable should be named var_tl and var_ad respectively. The underscore is omitted for local active variable. To summarized, active variables un, pun, zun will give un_tl, pun_tl and zun_tl in the tangent module*
- *For historical reasons, some modules do not follow these rules, feel free to update them.*

If the TAM module is a direct translation of the corresponding OPATAM routine, a good practice is to account for it in the first line of the history (i.i stating that the 8.2 version was done by e.g Huey, Dewey, and Louie)

2.4.2 Workflow for introducing additional developments

coming soon

2.5 Frequently asked questions

2.5.1 I can't use GDB after compiling with FCM

When I run the code with GDB for debugging purposes, the library lib__fcm__nemotam.a is not found, and thus gdb does not have access to symbolic informations associated with the modules of the code : we can't set breakpoints, we can't look for the content of variables, etc. This library is a temporary one generated by FCM during the build process, but it is removed once the compilation is finished.

This problem only appears on Mac OSX's computers. For unknown reasons, the linker looses track when going from fcm's temporary libraries to the final libraries. The temporary workaround is to recreate these temporary libraries from the object files (but you will have to redo it each time you compile):

```
cd $SCRATCH_POMME/fcmbuild/POMME_Z46_mac_intel.1.1/build/lib/
ar -q lib__fcm__nemotam.a ../obj/*.o
```

2.5.2 I have many commas in the namelist generated by piano

This happens if your config namelist have lines ending with a comma. This feature is accepted by FORTRAN, but the PIANO python script parser do not handle this. Removing the commas at the end of each line will solve the problem.

2.5.3 I have negative values in my cost function terms

See congrad.F90 :

```
Compute cost for diagnostic purposes only
pf=zf0+0.5*DOT_PRODUCT(...)
```

It is this term in congrad which appears to be < 0 and is written to the cost file, whereas we would expect to write the pcost from simvar.F90. *Under investigation*

2.5.4 My run stops with 'PROBLEM WITH THE MATRIX OR WITH THE PRECONDITIONER' on stderr

This problem was reported in the POMME and GYRE configurations.

First, it is worth noting that this message disappears if you set

```
ln_tst_grad = .TRUE.
```

which is absolutely not a normal behaviour. *Needs further investigation.*

The problem is solved by increasing the namelist nmin parameter. For example, change

```
   nmin        =    300    !  minimum of iterations for the SOR solver
```

into :

```
   nmin        =    500    !  minimum of iterations for the SOR solver
```

in namsol. This behaviour is not normal, it suggests a convergence problem in the tangent barotropic solver. *Needs further investigation.*

List of Figures

1.1 MPI_ALLGATHER task description (ref: `http://rc.usf.edu/tutorials/classes/tutorial/mpi/chapter8.html` 21

2.1 Code organization 23
2.2 Releationships between executable files (first column) and available tests (second column). The thrid column focuses on the main namelist's parameters ('*namtst*' for adjoint, gradient, tangent tests, '*namobs*' for single obs.experiement and '*namhlt*' for linear tangent hypothesis) 27
2.3 Tangent test type: Four kinds of test results. Top left (a), linear routine with a residual following the error machine. Top right (b), non-linear routine with a residual decreasing as p-order of 2. Bottom left (c), linear routine with a residual decreasing as p-order of 1. Bottom right (d), non-linear routine with a residual decreasing as p-order of 1. (Legend: Solid line is $N_p = M(x + p\delta x) - M(x)$, circle symbol $L_p = L(p\delta x)$, triangle symbol $N_p - L_p$) 29
2.4 **Tangent linear hypothsesis validty for *GYRE_Z31***: plot in function of time-window in days over 1 year. Top graphs are RMS of T, S, U and V variables in S.I units. Bottom graphs are related coefficient correlation. 33

List of Tables

1.1 Fields stored in the reference trajectory 9
1.2 List of modified declarations in the direct code NEMO 11
1.3 Hand coding status of NEMOTAM, Dynamics part 12
1.4 Hand coding status of NEMOTAM, Tracers and Surface Boundary Condition part . 13
1.5 Available option in NEMOTAM, part1 14
1.6 Available option in NEMOTAM, part2 15
1.7 Adjoint test labels definition 17
1.8 Tangent Linear tests. First-order check (a) and second-order check (b) 18
1.9 ORCA2 Results of Adjoint test fo sub-routines 21

2.1 ORCA2 Results of Tangent tests for sub-routines 30
2.2 ORCA2 Results of the Adjoint tests for the main sub-routines 31
2.3 ORCA2 Results of the Adjoint tests for the main routine *step_tam* for a run-window of one day. From line one to four, each control parameter is perturbed individually. Column six, all control parameter are perturbed. 32
2.4 ORCA2 Results of the Adjoint tests for the main routine *step_tam* for a run-window of five days. From line one to four, each control parameter is perturbed individually. Column six, all control parameter are perturbed. 32

APPENDIX : Acronyms

- AD : Automatic differentiation
- AGRIF : Adaptive Grid Refinement In Fortran. Outil de raffinement de maillage et de nesting développé au sein de MOISE et inclus dans NEMO.
- ESOPA : Equipe Système OPA, qui gérait les développements de OPA. A été remplacée par la NEMO-Team après la mise en place du consortium NEMO
- MITgcm : Modèle de circulation général pour l'océan et l'atmosphère développé au MIT
- NEMO : Nucleus for European Modelling of the Ocean.
- NEMOTAM : TAM pour NEMO
- NEMO-Team : Equipe gérant les développements de NEMO
- NEMOVAR : système d'assimilation variationnelle de données pour NEMO
- OPA : Ocean Parallélisé. Le modèle direct d'océan, composant de NEMO.
- OPATAM : TAM pour OPA 8.x
- OPAVAR : système d'assimilation variationnelle de donnés pour OPA 8.x
- ORCA : grille globale pour OPA
- PONGO : Pôle d'Océan Numérique GrenOblois, regroupant les équipes MOISE (LJK/INRIA) et MEOM (LEGI)
- ROMS : Regional Ocean Modelling System
- TAF : Transformation of Algorithms in Fortran. Générateur automatique de code tangent et adjoint utilisé pour MITgcm.
- TAM : Modèles Tangent et Adjoint
- TAPENADE : Générateur automatique de code tangent et adjoint développé par le projet TROPICS de l'INRIA